UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

PLGRAM

Presentado por Víctor Renuncio Tobar
en Universidad de Burgos — 16 de junio de 2016
Tutores: Dr. César Ignacio García Osorio y Álvar Arnaiz González

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

D. César Ignacio García Osorio, profesor del departamento de Ingeniería Civil, área de Lenguajes y Sistemas Informáticos y D. Álvar Arnaiz González, profesor del departamento de Ingeniería Civil, área de Lenguajes y Sistemas Informáticos

Exponen:

Que el alumno D. Víctor Renuncio Tobar, con DNI 71.264.303-E, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado PLGRAM.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección de los que suscriben, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 16 de junio de 2016

Vº. Bº. del Tutor:

D. Cesar Ignacio García Osorio

Vº. Bº. del co-tutor:

D. Álvar Arnaiz González

Resumen

Desarrollo de una aplicación de escritorio que de forma automática genere ejercicios de análisis sintáctico descendente ($LL(1)$) y ascendente ($SLR(1)$, $LALR(1)$, $LR(1)$) para publicarlos como cuestionarios en Moodle y en formato LaTeX para su uso en pruebas escritas.

Descriptores

Generación de cuestionarios, análisis sintáctico ascendente, análisis sintáctico descendente, $LL(1)$, $SLR(1)$, $LALR(1)$, $LR(1)$.

Abstract

Development of a desktop application that automatically generates down parsing ($LL(1)$) and rising ($SLR(1)$, $LALR(1)$, $LR(1)$) exercises to publish them as questionnaires in Moodle and LaTeX format for use in written tests.

Keywords

Questionnaire generation, bottom-up parsing, top-down parsing, $LL(1)$, $SLR(1)$, $LALR(1)$, $LR(1)$.

Índice general

Índice de figuras

Introducción

1.1. Introducción

Una de las principales características del actual concepto de docencia, sobre todo universitaria, es el gran auge que ha experimentado la docencia online. Esto se ve reflejado no sólo en el apoyo que supone a la docencia presencial, sino en la demanda de educación a distancia que cada vez tiene un mayor público.

Un claro ejemplo es el campus virtual de la Universidad de Burgos. UBUVirtual es una plataforma virtual basada en WebMoodle, que a su vez es una plataforma de eLearning utilizada por una gran cantidad de usuarios.

Por tanto, dotar de todas las herramientas necesarias a dicha plataforma para ofrecer un soporte útil y dinámico tanto al alumno como al profesorado, es una de las prioridades a tener en cuenta. Surge así la necesidad de realizar pruebas y exámenes a través de la plataforma.

En este proyecto se presenta una aplicación de escritorio que permite generar cuestionarios sobre análisis ascendente y descendente. Además dicha aplicación permite su posterior publicación en Moodle y en formato LaTeX para su futuro uso en pruebas escritas.

Objetivos del proyecto

2.1. Objetivos técnicos

Los principales objetivos técnicos perseguidos mediante el desarrollo de la aplicación han sido:

1. Construcción de una GUI (Interfaz Gráfica de Usuario) que permita visualizar de forma rápida la aplicación desarrollada.

2. Construcción de una CLI (Command Line Interface) que permita trabajar de forma rápida y directa con la aplicación desarrollada.

3. Procesado y manipulación de gramáticas mediante procesadores del lenguaje.

4. Obtención del $FIRST$ y del $FOLLOW$ de una gramática.

5. Obtención de la Tabla de Análisis Sintáctico Predictivo ($TASP$) de una gramática.

6. Obtención de los conjuntos de items $LR(0)$ y $LR(1)$.

7. Obtención de las tablas de análisis sintáctico *ACCIÓN e IR A* para análisis $SLR(1)$, $LR(1)$ y $LALR(1)$.

8. Exportación de cuestionarios en formato *XML* compatible con la plataforma Moodle.

9. Exportación de cuestionarios en formato LaTeX.

10. Creación de un manual de usuario detallando todas las opciones de la aplicación.

2.2. Objetivos académicos

Tal y como se indica en la guía docente del Trabajo Fin de Grado, algunos de los objetivos académicos que se pretenden alcanzar con el mismo son los siguientes:

1. Desarrollo de un trabajo personal donde se apliquen los conocimientos teóricos y prácticos adquiridos en la titulación.
2. Desarrollo de la capacidad creativa mediante el planteamiento y resolución de un problema real.
3. Uso y aprendizaje de nuevas herramientas y aplicaciones vinculados a la Ingeniería Informática.
4. Desarrollo de la capacidad de exposición en público y defensa y argumentación del trabajo realizado.
5. Comunicar correctamente en otro idioma un resumen coherente del trabajo realizado.

2.3. Objetivos personales

El uso de VersionOne[1] me ha permitido organizar y mantener un control sobre las tareas que debía realizar. He realizado sesiones (*sprint*) de una semana que han coincidido con las reuniones con los tutores.

El uso de GitHub[2] me ha permitido llevar el control de versiones desde el inicio del trabajo.

He podido profundizar en el funcionamiento de ciertos aspectos de la máquina virtual de Java: sus librerías nativas, sus métodos de carga dinámica de librerías, paquetes y clases y sus efectos específicos sobre procesos ya en ejecución.

He aprendido a aprovechar las ventajas del lenguaje de programación Java[3]: mantener su capacidad multiplataforma evitando realizar llamadas al sistema operativo, sacar partido de sus librerías nativas para la carga de objetos externos y lectura de objetos internos compilados y también explorar las posibilidades de librerías Java de licencia gratuita creadas por su inmensa comunidad de desarrolladores.

[1] http://www.VersionOne.com/
[2] http://www.GitHub.com/
[3] http://www.java.com.es/

Conceptos teóricos

El análisis sintáctico (*parser*) es la fase del procesador de lenguaje que toma como entrada los componentes léxicos (*tokens*) que le envía el analizador léxico y comprueba si con ellos se puede formar alguna sentencia válida del lenguaje.

La sintaxis de los lenguajes de programación habitualmente se describe mediante gramáticas libres de contexto (o gramáticas tipo 2).

Estas gramáticas están formadas por:

- Un conjunto finito y no vacío de símbolos terminales(o tokens): Σ.
- Un conjunto de símbolos no terminales: N.
- Un símbolo especial llamado axioma o símbolo inicial $S \in N$.
- Un conjunto de producciones que relacionan los símbolos terminales con los no terminales, de la forma: $X \to w$ con $X \in N$ y $w \in (\Sigma \cup N)^*$.

Para verificar si una instrucción pertenece al lenguaje, se construye el árbol sintáctico de la instrucción a partir de los tokens recibidos y que constituirán las hojas del árbol sintáctico.

Si el analizador sintáctico no puede formar una sentencia produce un mensaje de error, tratando de indicar las causas por las cuales no puede formar la sentencia.

Para realizar el análisis es necesario calcular previamente tres tipos de conjuntos: anulables, iniciales ($FIRST$) y seguidores ($FOLLOW$). Los anulables son los no terminales que pueden generar la palabra vacía. El conjunto $FIRST$ de una cadena es el conjunto de terminales que pueden comenzar cualquier cadena obtenida a partir de la de partida y el conjunto $FOLLOW$ de un no

terminal es el conjunto de terminales que pueden aparecer inmediatamente después del no terminal.

Para las gramáticas libres de contexto existen dos tipos básicos de analizadores sintácticos: los descendentes (*top-down*) y los ascendentes (*bottom-up*). Los ascendentes intentan construir el árbol desde las hojas hasta la raíz, mientras que los descendentes comienzan por la raíz y bajan hasta las hojas.

3.1. Análisis sintáctico descendente

Los métodos descendentes tratan de generar una forma sentencial a partir del axioma reconstruyendo una derivación más a la izquierda en orden directo: $S \Rightarrow_l^* w$. El árbol de derivación se construye desde la raíz hasta las hojas (*análisis LL(k)* o *top-douw*)

Un analizador de este tipo está formado por el buffer de entrada, una pila, una tabla de análisis y el programa de control del analizador *(PDA)*.

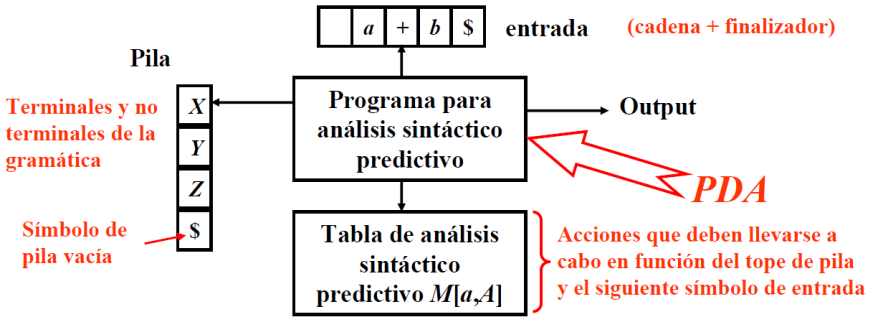

Figura 3.1: Estructura del analizador descendente.

El problema clave durante el análisis sintáctico predictivo es determinar la producción que debe aplicarse a un no terminal.

El analizador sintáctico no recursivo busca la producción que debe aplicarse en una tabla de análisis sintáctico predictivo $TASP$. A la vista del tope de pila y del símbolo nos dice que acción llevar a cabo.

Construcción de la *TASP*

Está compuesta por entradas de la forma $M[a, X] = (X \rightarrow w)$ cuando $a \in FIRST(w.FOLLOW(X))$.

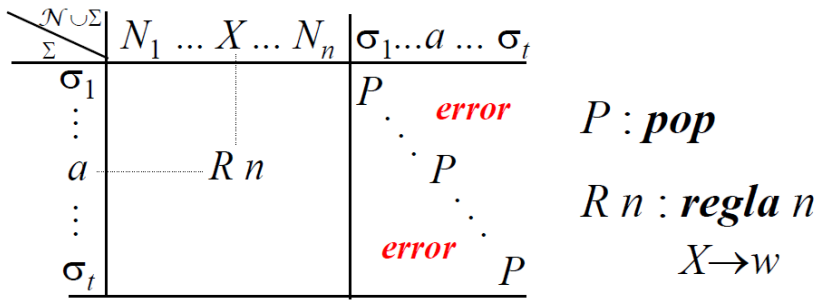

Figura 3.2: Tabla de análisis sintáctico predictivo

Uso de la *TASP*

Las acciones del PDA en función de la entrada y de la pila son:

Si $X = a = \$$, cadena reconocida y se finaliza el análisis.

Si $X = a \neq \$$, hacemos un pop en la pila y llamamos al analizador léxico para obtener un nuevo token.

Si $X \neq a$, se ha producido un error, llamar a la rutina de recuperación de errores y emitir un mensaje del tipo «se esperaba un X pero se encontró un a».

En cualquier otro caso se consulta la entrada de la tabla $M[a, X]$. Debe aparecer una producción de la forma $X \rightarrow Y_1, Y_2 \ldots Y_n$. Se hace un *pop* de X y se reemplaza por $Y_1, Y_2 \ldots Y_n$ con Y_1 en el tope de la pila. Si la entrada de tabla está vacía se llama a la rutina de recuperación de error y se emite un mensaje como «X no puede comenzar por a».

Ejemplo

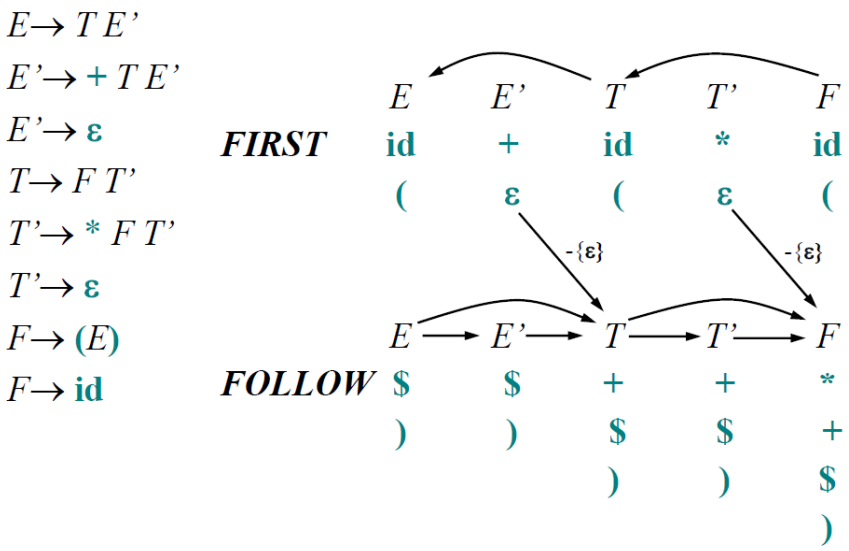

Figura 3.3: Gramática con $FIRST$ y $FOLLOW$

No Terminal	Símbolo de Entrada					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

Figura 3.4: $TASP$ de la gramática

PILA	ENTRADA	SALIDA
$E	id + id * id$	$E \rightarrow TE'$
$E'T	id + id * id$	$T \rightarrow FT'$
$E'T'F	id + id * id$	$F \rightarrow id$
$E'T'id	id + id * id$	
$E'T'	+ id * id$	$T' \rightarrow \varepsilon$
$E'	+ id * id$	$E' \rightarrow +TE'$
$E'T+	+ id * id$	
$E'T	id * id$	$T \rightarrow FT'$
$E'T'F	id * id$	$F \rightarrow id$
$E'T'id	id * id$	
$E'T'	* id$	$T' \rightarrow *FT'$
$E'T'F*	* id$	
$E'T'F	id$	$F \rightarrow id$
$E'T'id	id$	
$E'T'	$	$T' \rightarrow \varepsilon$
$E'	$	$E' \rightarrow \varepsilon$
$	$	

Figura 3.5: Traza de la cadena "$id + id * id$$"

Gramáticas $LL(1)$

Una gramática cuya *TASP* no tiene entradas múltiples se dice que es $LL(1)$.

Si una gramática es $LL(1)$ entonces es no ambigua, es no recursiva a izquierdas y debe estar factorizada. Sin embargo las afirmaciones inversas no son ciertas.

Se pueden modificar las gramáticas, factorizando por la izquierda o eliminando la recursividad a izquierdas sin modificar el lenguaje que se está reconociendo.

Las gramáticas $LL(1)$ permiten construir de forma automática una analizador determinista con tan solo examinar en cada momento el símbolo actual de la cadena de entrada para saber qué producción aplicar.

3.2. Análisis sintáctico ascendente

La idea es generar una forma sentencial a partir del axioma, reconstruyendo una derivación más a la derecha en orden inverso: $S \Rightarrow_r^* w$.

Los más utilizados son los analizadores sintácticos LR que utilizan la técnica de análisis por desplazamiento/reducción. Se parte de la cadena a analizar y se trata de reducir hasta llegar al símbolo inicial de la gramática. En cada paso de reducción se sustituye una subcadena determinada que concuerde con el lado derecho de una producción por el símbolo del lado izquierdo de dicha producción.

Si en cada paso se elige correctamente la cadena, se consigue una derivación por la derecha en sentido inverso.

Las tres técnicas más comunes para construir tablas de análisis LR son:

- SLR ($LRsimple$): Es la más fácil de implementar, pero la menos potente de las tres. Solo son capaces de reconocer un número reducido de gramáticas.
- LR *canónico*: Es la más potente y la más costosa de las tres.
- $LALR$ ($LookaheadLR$): Tiene un costo y una potencia intermedia entre los dos anteriores.

Un analizador sintáctico LR consta de un buffer de entrada, una pila, un programa conductor y una tabla de análisis sintáctico con dos partes, la de $ACCIÓN$ y la de $IR\ A$.

El programa conductor es el mismo para todos los analizadores sintácticos LR, solo cambian la forma de obtener las tablas de un tipo de análisis a otro.

El programa utiliza una pila para almacenar una cadena de la forma $s_0X_1s_1X_2s_2 \ldots X_ms_m$ donde s_m está en la cima. Cada X_i es un símbolo gramatical y cada s_i es un símbolo llamado estado.

Cada símbolo de estado resume la información contenida debajo de él en la pila, y se usan la combinación del símbolo de estado en la cima de la pila y el símbolo en curso de la entrada para indexar la tabla del análisis sintáctico y determinar la decisión de desplazamiento o reducción del analizador.

Poner gráfica

En cada paso hay cuatro opciones:

- Poner el token actual en la cima de la pila y llamar al analizador léxico para obtener un nuevo token ($SHIFT$).

- Decidir que los símbolos en la cima de la pila forman un asidero, y entonces desapilarlos y reemplazarlos por la parte izquierda de su producción ($REDUCE$).
- Finalizar aceptando la cadena de entrada.
- Finalizar señalando un error.

Al realizar análisis ascendente pueden surgir conflictos. Ocurre un conflicto cuando en una entrada de la tabla hay más de una acción. Hay dos tipos de conflicto $SHIFT - REDUCE$ y $REDUCE - REDUCE$. En algunos casos se pueden resolver mirando el siguiente símbolo de la entrada.

Gramáticas LR

Una gramática para la que se puede construir una tabla de análisis LR se dice que es una gramática LR. Una gramática que puede ser analizada por un analizador LR mirando hasta k símbolos de entrada por delante (*lookaheads*) en cada movimiento, se dice que es una gramática $LR(k)$.

Análisis $SLR(1)$

Es la técnica más sencilla que permite un análisis LR con previsión de *LookAhead* de un símbolo.

Se basa en la construcción de los conjuntos de items $LR(0)$ de la gramática Un item $LR(0)$ de una gramática G es una de sus producciones, con un punto en algún lugar de su parte derecha. El 0 indica que no utilizamos *lookaheads*. Los ítems se van a agrupar en estados de un autómata capaz de reconocer asideros en la cima de la pila.

Ante un conflicto $SHIFT - REDUCE$: un item $[A \to \alpha_1 \bullet X\alpha_2]$ y otro $[B \to \delta\bullet]$ en el mismo conjunto, se calcula el $FOLLOW(B)$, si no incluye el símbolo de detrás de la marca en el item $[A \to \alpha_1 \bullet X\alpha_2]$ se puede resolver el conflicto.

Ante un conflicto $REDUCE - REDUCE$: $[A \to \delta\bullet]$ y $[B \to \gamma\bullet]$ en el mismo conjunto, si son disjuntos $FOLLOW(A)$ y $FOLLOW(B)$ puede resolverse.($S \Rightarrow^* \phi At \Rightarrow \phi\delta t$ pero $S \Rightarrow^* \phi Bu \Rightarrow \phi\gamma u$).

Análisis $LR(1)$

La construcción de item $LR(1)$ es muy parecida a la de items $LR(0)$.

En el inicio se hace igual: $[S' \to \bullet S\#, \#] \in Set(0)$. La que cambia es la operación de cierre. Cierre: Completar el núcleo de un conjunto de items. Si $[A \to x \bullet By, u] \in Set(n) \wedge B \in N \wedge B \to w \in P \Rightarrow [B \to \bullet w, v] \in Set(n) \forall v \in FIRST(yu)$.

1 $G \to S\#$
2 $S \to CC$
3 $C \to cC$
4 $C \to d$

Estado	Acción			Goto	
	c	*d*	#	*S*	*C*
0	*d*3	*d*4		1	2
1			*acep*		
2	*d*6	*d*7			5
3	*d*3	*d*4			8
4	*r*4	*r*4			
5			*r*2		
6	*d*6	*d*7			9
7			*r*4		
8	*r*3	*r*3			
9			*r*3		

Figura 3.6: Ejemplo de gramática $LR(1)$ con tabla de *ACCIÓN* e *IR A*

Análisis $LALR(1)$

El análisis $LR(1)$ canónico requiere muchos más estados que el análisis $SLR(1)$. En lenguajes de programación reales la diferencia puede ser un factor de 10. Por el contrario hay algunas construcciones de los lenguajes de programación que dan problemas en el análisis $SLR(1)$.

El análisis $LALR(1)$ combina las ventajas de los dos análisis. Puede tratar con las gramáticas problemáticas para el análisis $SLR(1)$, y requiere menos estados que en el análisis $LR(1)$ *canónico*.

Una vez obtenidos los conjuntos $LR(1)$ se identifican aquellos en los que los items del núcleo sólo difieren en los lookahead y se fusionan, haciendo que su lookahead sea la unión. Sobre el nuevo conjunto de conjuntos de items marcados se aplica el mismo algoritmo de obtención de tablas *ACCIÓN* e *IR A* que para el $LR(1)$ *canónico*. Pueden surgir conflictos REDUCE/REDUCE que antes no existían en un conjunto C si $[X \to \alpha\bullet, l_1] \in C \wedge [Y \to \beta\bullet, l_2] \in C \wedge l_1 \cap l_2 \neq \emptyset$. Si surgen conflictos, la gramática no es $LALR(1)$.

Técnicas y herramientas

4.1. Técnicas

SCRUM

SCRUM[4] es una metodología ágil en la que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente y obtener el mejor resultado posible de un proyecto.

Se basa en la creación de procesos iterativos e incrementales, así se pueden realizar estimaciones de rendimiento futuro sobre las cuales se pueden tomar decisiones y controlar los riesgos.

UML

Lenguaje Unificado de Modelado (Unified Modeling Language)[5] es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad en el diseño de aplicaciones Orientadas a Objetos.

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

4.2. Herramientas

Todas las herramientas utilizadas para la elaboración del proyecto son gratuitas o se pueden obtener versiones de prueba en Internet.

[4] https://www.scrum.org//
[5] http://www.uml.org/

Java

Java[6] es un lenguaje de programación de propósito general, concurrente y orientado a objetos.

Se trata de un lenguaje fuerte y estáticamente tipado.

Distingue entre errores en tiempo de compilación y errores en tiempo de ejecución.

Java es un lenguaje de alto nivel, en el sentido de que los detalles de representación no son accesibles al programador. Incluye una administración automática del almacenamiento de datos en forma del recolector de basura, diseñado para evitar los problemas relacionados con la liberación manual de memoria. Es uno de los lenguajes de programación mas usados en el mundo.

Java 8

Java SE 8 es la edición más reciente del lenguaje de programación Java, y representa la mayor evolución del mismo en su historia.

Incluye nuevas características, mejoras y correcciones de bugs para mejorar la eficacia en el desarrollo y la ejecución de programas Java.

Las librerías de la plataforma Java mantienen una evolución paralela a la del lenguaje.

Eclipse

Eclipse[7] es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para el desarrollo integrado (IDE) de aplicaciones Java y una base para productos basados en Eclipse Platform.

JUnit 4

JUnit[8] es un sistema software utilizado para realizar pruebas sobre código Java, formando parte de la familia de herramientas de pruebas xUnit. Es una de las librerías Java más utilizadas en proyectos de código abierto.

La versión 4 de JUnit extiende y simplifica la funcionalidad de anteriores versiones, haciendo un uso extensivo del sistema de anotaciones de Java.

[6] https://www.java.com.es
[7] https://eclipse.org/
[8] http://junit.org/

Apache Maven

Maven[9] es una herramienta de administración y construcción de proyectos usada principalmente con Java.

Su funcionamiento se basa en la existencia de un fichero de configuración XML, el POM (Project Object Model). Este fichero define la construcción, emisión, documentación, empaquetado, pruebas, manejo de dependencias y múltiples otras tareas de manera centralizada.

JavaCC

JavaCC[10] (*Java Compiler Compiler*) es un generador de analizadores sintácticos Java. Funciona mediante la especificación de una gramática en un formato propio. A partir de este fichero, la herramienta genera un programa Java capaz de procesar texto y reconocer coincidencias con la gramática.

Además del generador en sí, JavaCC proporciona una serie de herramientas relacionadas, como un constructor de árboles (JJTree) y un informe en caso de error de los mejores entre los analizadores sintácticos disponibles.

VersionOne

VersionOne[11] es un sistema de control de versiones, diseñado para trabajar con proyectos de cualquier tamaño. VersionOne permite mantener múltiples ramas locales independientes, cuya creación, modificación y combinado resulta poco costoso. Esto permite aislar tareas, trabajando e incorporando cada una de manera totalmente separada.

Github

Github[12] proporciona un servicio de almacenamiento de repositorios remotos y un entorno de colaboración para desarrolladores.

Github proporciona herramientas de seguimiento de proyectos, incluyendo una wiki y un sistema de seguimiento de problemas (*issue tracker*) por repositorio. También es compatible con otras aplicaciones web, como por ejemplo Pivotal Tracker, que facilitan su integración en el proceso de desarrollo.

Enlace al repositorio de este proyecto: https://github.com/vrt0004/Trabajo

[9] http://maven.apache.org/
[10] https://javacc.java.net/
[11] https://www.versionone.com/
[12] https://github.com/

Moodle

Moodle (*Modular Object-Oriented Dynamic Learning Environment*) es una plataforma de enseñanza virtual (*e-learning*) desarrollada como software libre.

Es una aplicación para crear y gestionar plataformas educativas, espacios donde un centro educativo, institución o empresa, gestiona recursos proporcionados por sus docentes, organiza el acceso a los mismos y permite la comunicación entre todos los implicados (alumnado y profesorado).

Una de las características de Moodle es su capacidad para importar cuestionarios a partir de varios formatos, incluyendo texto plano, el formato propietario Gift y a partir de documentos XML.

LaTeX

LaTeX[13] es un lenguaje de marcadores para la preparación de documentos comúnmente usado en publicaciones técnicas o científicas.

Es un procesador de texto «*What you see is not what you get*», lo cual significa que lo que vemos durante la edición no es el documento final, sino las instrucciones para generarlo. Esto permite separar casi completamente el contenido del documento de su formato. La salida obtenida será la misma con independencia del dispositivo o sistema operativo empleado para su visualización o impresión.

TeXMaker

TeXMaker[14] es un editor libre de látex, moderno y multiplataforma para Linux , MacOSX y ventanas sistemas que integra muchas herramientas necesarias para desarrollar documentos con LaTeX, en una sola aplicación.

Texmaker incluye soporte Unicode, la corrección ortográfica, auto-completado, plegado de código y un visor de pdf con el apoyo SyncTeX y el modo de visualización continua. Texmaker es fácil de usar y configurar. Se distribuye bajo la licencia GPL.

MiKTeX

MiKTeX es una distribución TeX/LaTeX libre de código abierto para Windows. Una de sus características es la capacidad que tiene para instalar paquetes automáticamente sin necesidad de intervención del usuario. Al contrario que otras distribuciones, su instalación es muy sencilla.

[13] https://latex-project.org/
[14] http://www.xm1math.net/texmaker/

Mustache

Mustache[15] es un sistema de plantillas de lógica descendente para HTML, XML y otros muchos.

Se llama de lógica descendente porque no tiene comandos if, else, ni bucles for. En cambio, sólo se trabaja con etiquetas. Algunas etiquetas son reemplazadas por un valor, por un conjunto de valores o por nada.

XML

XML (*Extensive Markup Language*) es un lenguaje de etiquetas que se utiliza para crear documentos estructurados, compuestos de entidades que contienen en su interior datos u otras entidades. El estándar fue producido y es desarrollado por el *World Wide Web Consortium*[16].

Podemos verificar que un documento *XML* tiene el formato correcto validándolo contra un lenguaje de definición de esquemas: DTD[17],(*Document Type Definition*), *XML Schema*[18], RELAX NG[19]... También podemos verificar que un documento está 'bien formado', es decir, que cumple una serie de reglas gramaticales mínimas.

Commons CLI

La biblioteca Apache Commons CLI[20] proporciona una API para analizar las opciones de línea de comandos pasados a los programas.

También es capaz de imprimir mensajes de ayuda que detallan las opciones disponibles para una herramienta de línea de comandos.

ObjectAid UML Explorer

ObjectAid UML Explorer[21] es un complemento de visualización de código para eclipse. Permite mostrar el código fuente de un proyecto Java en forma de diagramas UML, reflejando el estado y las relaciones en el código, y actualizándose a medida que el código cambia.

---

[15] http://mustache.github.io/mustache.1.html
[16] http://www.w3.org/
[17] http://es.wikipedia.org
[18] http://www.w3.org/TR/xmlschema-0/
[19] http://www.relaxng.org/
[20] http://commons.apache.org/proper/commons-cli/
[21] http://www.objectaid.com/

EclEmma

EclEmma[22] es una herramienta que permite examinar la cobertura de pruebas en un proyecto Java. Dispone un un complemento para Eclipse que permite realizar las comprobaciones directamente desde el IDE.

[22] http://www.eclemma.org/

Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

Trabajos relacionados

Para comenzar a trabajar en el desarrollo de la presente aplicación se buscó información relacionada con la aplicación que se quería llevar a cabo.

Para ello, se realizó una investigación a través de la Web sobre aplicaciones que tuvieran una función similar a la desarrollada.

Aunque no existe una gran variedad de programas que realicen tareas similares a *PLGRAM*, entre los dos programas recomendados por el tutor se ha encontrado información suficiente para poder llevar a cabo la aplicación. Dichos programas son *PLQUIZ* y *BURGRAM*.

Dado que ambos programas han sido desarrollados por alumnos de la Universidad de Burgos, como trabajos de final de grado, el tutor pudo facilitarme su estructura y toda la documentación que estimó necesaria para la correcta ejecución de *PLGRAM*.

6.1. PLQUIZ

PLQUIZ[23] es una herramienta de escritorio que permite generar preguntas de test aleatorias (tipo quiz, cloze, de texto libre...) sobre problemas de algoritmos de análisis léxico. El formato utilizado para generar las preguntas es .XML para importarse a entornos virtuales de aprendizaje (Moodle), y .TEX para su impresión en papel a través de la obtención de código LaTeX.

Se han seguido sus pasos en el formato de la interfaz gráfica, pues tiene una organización muy clara para el usuario, permitiendo acceder rápidamente a las opciones disponibles.

[23]Roberto Izquierdo Amo, Julio 2014, Universidad de Burgos

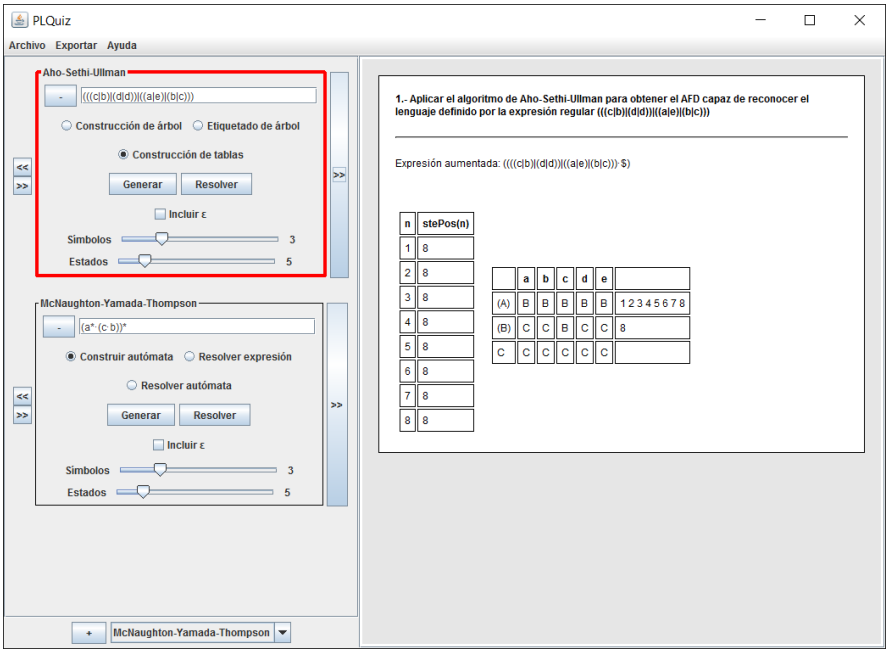

Figura 6.7: PLQUIZ

6.2. BURGRAM

BURGRAM[24] es una herramienta que permite seguir paso a paso el proceso de los distintos modos de análisis sintáctico (tanto ascendente como descendente). La aplicación permite editar gramáticas, visualizar el proceso completo de operación del algoritmo sobre las mismas y la generación de informes con los resultados.

Se han reutilizado varias de las clases de este programa pues realizan el análisis que la aplicación desarrollada necesitaba para obtener los datos que posteriormente se han tratado para obtener los ficheros .XML.

[24]Carlos Gómez Palacios, Enero 2008, Universidad de Burgos

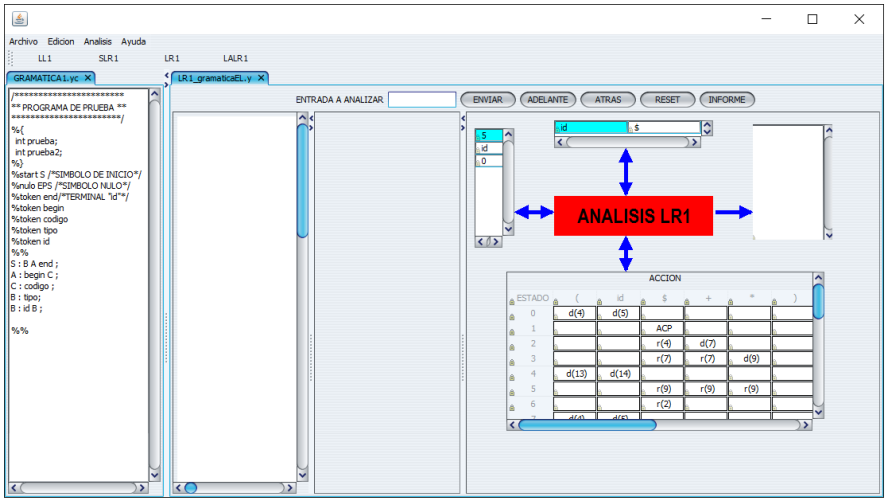

Figura 6.8: BURGRAM

6.3. SEFALAS

SEFALAS[25] es una herramienta que muestra el transcurso de la fase de análisis léxico y de analisis sintáctico. Es de bastante utilidad dado que existen multitud de estrategias de análisis sintáctico y, por regla general, puede provocar confusión.

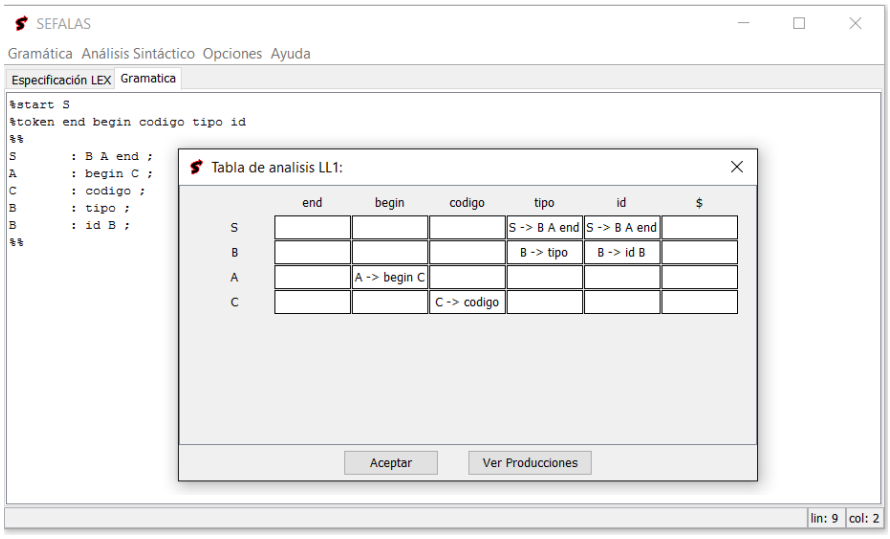

Figura 6.9: SEFALAS

[25] José Francisco Jódar Reyes, 2004, Universidad de Granada

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.